ESD-TR-71-346

MTR-2115

A GUIDE TO THE POTENTIAL USE OF SIMSCRIPT

P. R. Burleson

SEPTEMBER 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-71-C-0002

ESD-TR-71-346 MTR-2115

A GUIDE TO THE POTENTIAL USE OF SIMSCRIPT

P. R. Burleson

SEPTEMBER 1971

Prepared for

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

Project 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract F19(628)-71-C-0002

FOREWORD

This report presents the results of a study conducted by The MITRE Corporation, Bedford, Mass., under Contract No. F19(628)-71-C-0002, MITRE Project 5720. ESD program monitor is Mr. William J. Letendre, Technology Application Division, Directorate of Systems Design and Development. Publication of this report does not constitute Air Force approval of report's findings or conclusions. It is published only for the exchange and stimulation of ideas.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.

for P. R. Peckham
EDMUND P. GAINES, JR., Colonel, USAF
Director, Systems Design & Development
Deputy for Command and Management Systems

ABSTRACT

This report (1) identifies the features which distinguish SIMSCRIPT from general programming languages, permitting readers to judge for themselves the benefits of using SIMSCRIPT in their own applications; (2) outlines the language and implementation differences between the various versions of SIMSCRIPT; (3) specifies the resource requirements and relative advantages of implementing each version of SIMSCRIPT at MITRE/ESD; and (4) investigates the desirability of using SIMSCRIPT at ESD for analyzing problems related to computer performance.

ACKNOWLEDGMENTS

The author would like to recognize the assistance given by Miss J. C. DesRoches, J. H. McIntosh, and J. P. Hogan, of The MITRE Corporation, who reviewed this paper and contributed to both its form and content.

TABLE OF CONTENTS

LIST OF TABLES

SECTION I

INTRODUCTION

"A programmer is greatly influenced by the language in which he writes his programs; there is an overwhelming tendency to prefer constructions which are simplest in that language..."[1]

Increasing demands have been placed upon the Electronic Systems Division of USAF to provide support to Air Force users in simulating automatic data processing equipment (ADPE) systems performance. In the past, these demands were met with a technology base that included one simulation package and considerable reliance upon commercially contracted support for its use. One of the purposes of Project 5720 is to assist ESD in keeping abreast of technology in the ADPE simulation area, and to help provide this technology as the needs dictate.

A previous survey[2] identified SIMSCRIPT, which is a computer programming language oriented toward simulation, as a prime candidate for use in ADPE simulation. This report examines SIMSCRIPT in more detail, both as a general simulation tool useful to MITRE/ESD at large, and for its usefulness in ADPE simulation. The particular purposes of this report are:

(1) To identify the features which distinguish SIMSCRIPT from general programming languages, permitting readers to judge for themselves the benefits of using SIMSCRIPT in their own applications.

(2) To outline the language and implementation differences between the various versions of SIMSCRIPT.

(3) To specify the resource requirements and relative advantages of implementing each version of SIMSCRIPT at MITRE/ESD.

(4) To investigate the desirability of using SIMSCRIPT at ESD for analyzing problems related to computer performance.

It is not possible to accomplish (1) and particularly (2) above without discussing SIMSCRIPT features at the language level. Readers

unfamiliar with programming may find Sections III through V rather incomprehensible for that reason. In any case, the SIMSCRIPT language is extensive and powerful, and a full appreciation for its capabilities cannot be acquired without making the effort to learn detailed language provisions.

The remainder of this report is organized as follows: Section II outlines the historical development of SIMSCRIPT, Section III presents the language features which are oriented toward simulation, Section IV contrasts the language design of SIMSCRIPT I with that of SIMSCRIPT II, Section V covers differences between all language versions which result from the implementation rather than the language specification, and Section VI offers some generalizations from preceding sections together with considerations of efficiency and economy to reach conclusions for purposes (3) and (4) above.

SECTION II

SIMSCRIPT DEVELOPMENT

In 1963, Harry Markowitz, Bernard Hausner and Herbert Karr of The RAND Corporation published SIMSCRIPT, A Simulation Programming Language, which reported the design of a language specifically oriented toward systems simulation. It was the culmination of about three years effort on SIMSCRIPT, plus previous design experience with SPS-1 (Simulation Programming System-1) at RAND and GEMS (General Electric Manufacturing Simulator) by Markowitz at GE. Markowitz acted as chairman of the design team and had ultimate responsibility for the logical design of the system. Karr wrote the original documentation. The language was implemented by Hausner for the IBM 7040/7090, through translating SIMSCRIPT program statements into FORTRAN and passing this text to the FORTRAN compiler. This implementation of the language is now referred to as SIMSCRIPT I, and has become relatively obsolete.

Markowitz and Karr left RAND some time after 1963 to set up California Analysis Center, Inc. (CACI), a firm which markets a version of SIMSCRIPT under the nomenclature I.5 (read "eye" point five). SIMSCRIPT I.5 is substantially identical to its predecessor, except that program statements are assembled directly into machine code. SIMSCRIPT I programs will therefore compile and execute under the SIMSCRIPT I.5 system as long as they contain neither FORTRAN inserts nor LOAD, RECORD, or RESTORE statements.[1] SIMSCRIPT I.5's language differences therefore consist primarily of increasing the power of and relaxing restrictions for a subset of instructions. SIMSCRIPT I.5 has been implemented on: IBM 7040/44, 7090/94, and 360, NCR 200, CDC 3600/3800, CDC 6400/6500/6600, Philco 210/211/212, UNIVAC 490/494/1107/1108, RCA Spectra 70/45 and above and GE 625/635. With the exception of the GE compiler, which was done by Digitek, Inc. rather than C.A.C.I., all I.5 compilers are claimed to be completely compatible. Users can therefore utilize old programs on new equipment. Since SIMSCRIPT is most frequently used for simulation, and simulation models are seldom applicable beyond the system they were designed to replicate, this transferability advantage reduces to one of minimizing phaseover problems for current simulation efforts.

Almost as soon as work on SIMSCRIPT I was completed, an effort to produce an improved version of the language was initiated.

---

[1] These statements reference I/O operations and were not considered part of the SIMSCRIPT I.5 language. They were, however, incorporated into the SIMSCRIPT I language.

Markowitz again directed the design of the language and Hausner worked on the implementation. The language was eventually produced in 1968 by Philip Kiviat and Richard Villanueva and reported in a RAND document.[3] SIMSCRIPT II represents a major departure from previous versions since it provides a truly general programming language that can be used in a wide variety of applications. The features of the language which are oriented toward simulation utilize the same data structures and event mechanisms as previous SIMSCRIPT versions, but the syntax and semantics utilized to specify these constructs are significantly different. The nature and import of these differences will be discussed in succeeding sections.

The RAND report already referenced, which was also issued in book form by Prentice Hall,[4] is a very lucid exposition of a rather difficult subject, a language that is both rich (flexible) and powerful. Unlike the SIMSCRIPT I documentation, which consists of a single 140 page volume organized in reference manual format, the SIMSCRIPT II reports include the text already mentioned (380 pages),[5] which is designed as a teaching vehicle, a language reference manual, listing instruction syntax, and an implementation manual[6] which identifies the interfaces between SIMSCRIPT and system hardware and software.

The text is divided into five chapters which are identified as "levels" of the language as follows:[3]

"Level 1: A simple teaching language designed to introduce programming concepts to nonprogrammers.

Level 2: A language roughly comparable in power with FORTRAN but departing greatly from it in specific features.

Level 3: A language roughly comparable in power to ALGOL or PL/I, but again with many specific differences.

Level 4: That part of SIMSCRIPT II that contains the entity - attribute - set features of SIMSCRIPT. These features have been updated and augmented to provide a more powerful list - processing capability. This level also contains a number of new data types and programming features.

Level 5: The simulation-oriented part of SIMSCRIPT II containing statements for time advance, event-processing, generation of statistical variates, and accumulation and analysis of simulation-generated data."

This division into levels is solely for expositional purposes and corresponds to no real ordering or structure within the language itself.

The language was implemented at RAND for the IBM 360/65 under version 15/16 of MVT. At the time of program submission to the SHARE library (1969) there were "no known restrictions on using the compiler under PCP or MFT II, and . . . no known OS release dependencies." (6) Unfortunately time and events proved otherwise, and an attempt by Villanueva to resubmit the program to SHARE was refused,[2] since IBM no longer supports the library. Several installations[3] have gotten SIMSCRIPT II running under MVT but only recently (March 1971) has RAND made available a working MFT version. A number of statements defined for the language were not implemented in this version. Compilation is achieved through translation to assembly language and use of the IBM assembler.

In 1969, Kiviat and Villanueva left RAND to form Simulation Associates (S/A) with Henry Kleine, Arnold Ockene, and later Robert Parente. The company developed a faster version of SIMSCRIPT II, that permits more statement types than the RAND implementation. This latest version, called SIMSCRIPT II Plus, has been marketed by S/A. Despite the technical excellence of the language and its implementation,[4] Simulation Associates ceased existence as a business entity in March 1971. The assets of the company have been purchased by C.A.C.I., who have recently announced that they are offering a slightly modified version of II Plus under the name SIMSCRIPT II.5.

The present status of the languages/implementations therefore is: (1) the SIMSCRIPT I language and implementation are inseparable, as both were reported together, (2) I.5 uses the language design of SIMSCRIPT I, with few modifications, (3) the language design specifications for SIMSCRIPT II are reported in the RAND and Prentice-Hall publications, and (4) neither the RAND implementation of SIMSCRIPT II nor S/A's II Plus nor C.A.C.I.'s II.5 have yet incorporated the full repertoire of statements defined for the language.

---

[2] Apparently COSMIC will now accept new or revised programs. RAND is considering the submission to SHARE of a revised version of SIMSCRIPT II which will operate under MFT.

[3] Yale, Princeton, Columbia, and others.

[4] For example, although Yale had a working version of the free SIMSCRIPT II, they eventually purchased S/A's implementation.

SECTION III

SIMSCRIPT'S SIMULATION AIDS

The intent of SIMSCRIPT's originators was that the special simulation-oriented features of the language would provide the mechanisms common to most simulation exercises, thus reducing programming time and easing the modification of models. Since both versions I and II of the language possess the same data structures and mechanisms for modifying data structures (called "world view"), and since it is precisely these features that distinguish SIMSCRIPT from general purpose programming languages, a fairly brief identification of the SIMSCRIPT world view is provided below.

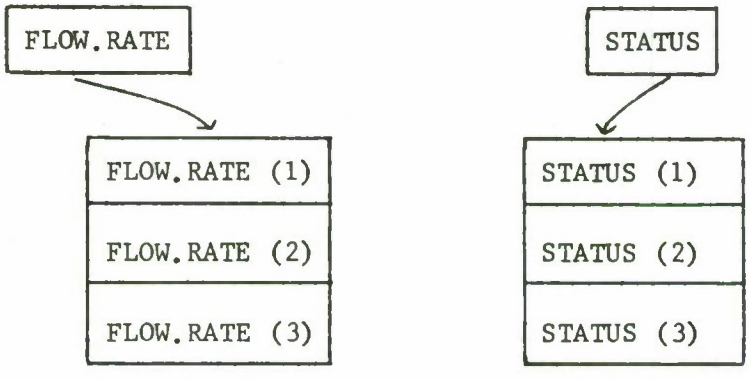
DATA STRUCTURES

Every simulation model consists of two prime components: (1) a description of the interrelationships between system elements that define system state at a point in time, and (2) a specification of the ways in which system state can change. In SIMSCRIPT, system elements are called "entities." Entities are the significant objects present in the system modeled. For example, the simulation of a gas station would probably define as entities the various pumps, attendants, and cars which interact to produce the behavior of interest. The description of an entity is done through associating "attributes" with it, whose values define a particular configuration or state of the entity. Entities may be "temporary" i.e. created and/or destroyed during the simulation, or "permanent," meaning just that. The distinction between permanent and temporary entities is made primarily for the purpose of computational efficiency, since the same static descriptors and change mechanisms can be applied to them. Entities can be classified into sets, in which they can be ranked on FIFO, LIFO,[5] or attribute value bases.

The means of implementing these data structure constructs are rather interesting. Definitional statements specify the data structure to the compiler, and core storage is reserved for entities on a dynamic basis at execution time. Thus if gas pumps were permanent entities, in the beginning of a SIMSCRIPT program, GAS.PUMP would be defined as a class of permanent entities with specified attributes (e.g. FLOW.RATE and a STATUS descriptor). This definition would set up "pointers" called FLOW.RATE and STATUS which initially (i.e. after the program is loaded into core) contain zero. During program execution (usually during initialization) a value specifying the number of gas pumps is read, a statement creating every gas pump

[5] FIFO means first-in first-out, and LIFO last-in, first out.

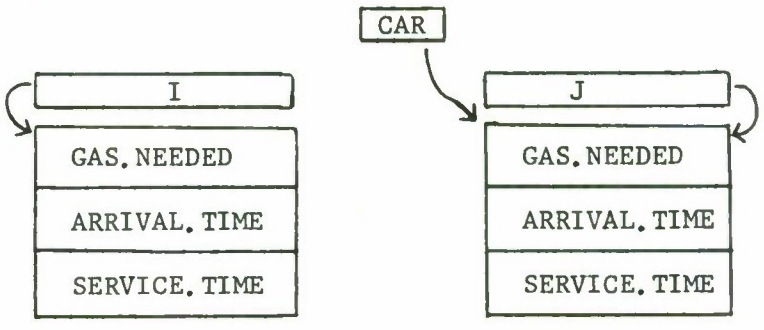
(reserving storage) is executed, and values are assigned to attributes, usually through reading data. After this is complete, the pointers FLOW.RATE and STATUS point to lists in core where the attributes of the gas pumps are stored. If the number of pumps was 3, the structure generated looks like:



Once generated, the only way to free the core storage occupied by permanent entities is to "destroy" all permanent entities of the same type (e.g. all GAS.PUMPs). A global variable called GAS.PUMP will be defined automatically by the system, and it contains an integer identifying the particular GAS.PUMP referenced last.

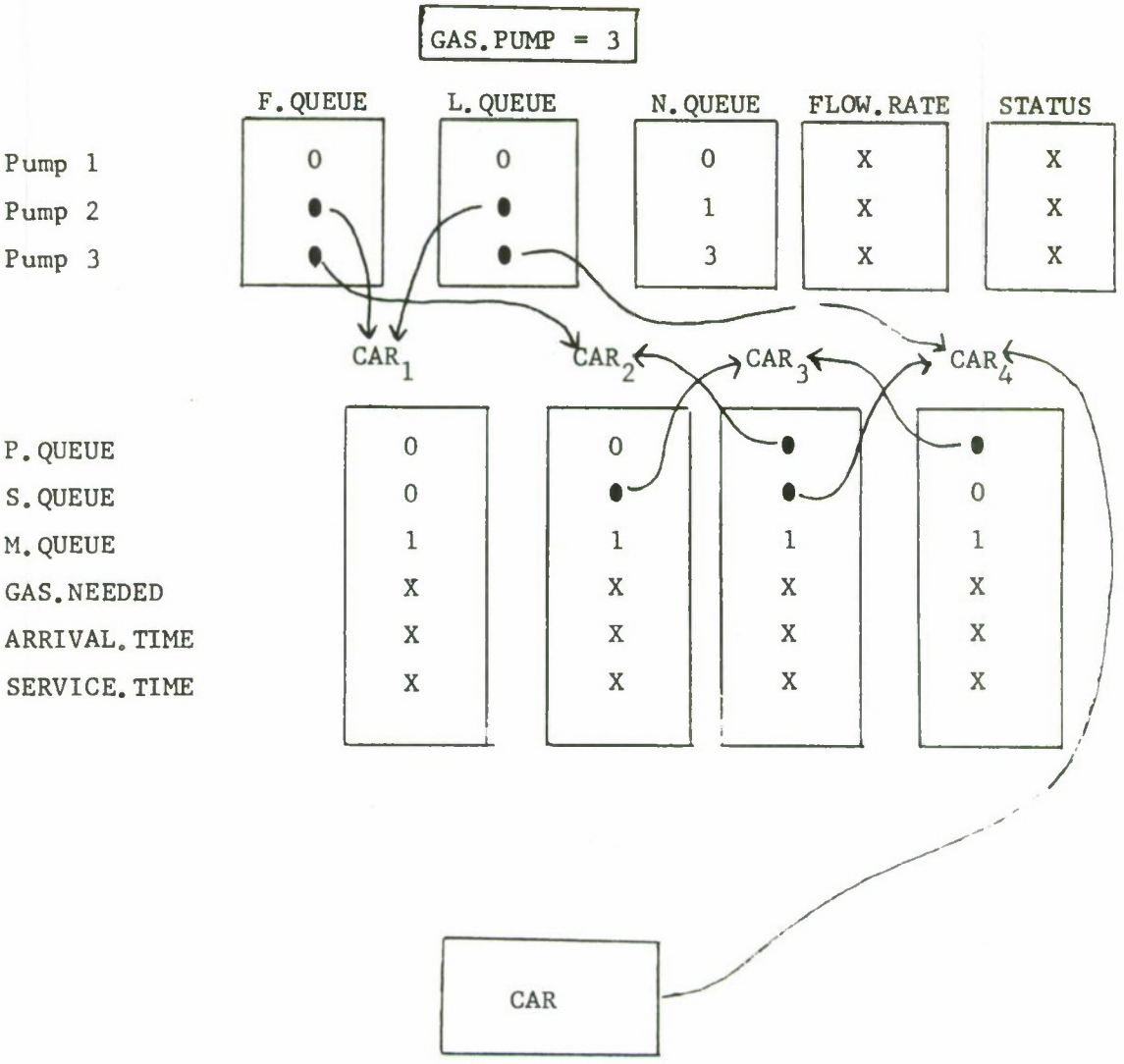
Attendants would probably be modeled as permanent entities also. Cars would be modeled as temporary entities, since they enter the system (gas station), are serviced, and depart. Let CAR have the attributes GAS.NEEDED, ARRIVAL.TIME and SERVICE.TIME, which again are specified in the beginning of the program by a non-executable definition. This definition results in a single pointer, a global variable called CAR which initially contains a zero.

During the execution of the program, the statements "CREATE A CAR CALLED I" followed by "CREATE A CAR CALLED J" produce the following structure:



The symbol I can be used to reference a pointer to the storage reserved for the attributes of I. In order to avoid referencing of temporary and permanent entities by name (e.g. I), global variables are defined with the same name as the entity class. Thus "CREATE A CAR" is acceptable, and equivalent to "CREATE A CAR CALLED CAR". The global variable "CAR" always points to the most recently referenced car, and is updated automatically when cars are filed into or removed from sets. Therefore, reference to GAS.NEEDED(J) is equivalent to GAS.NEEDED, as long as the pointer in the global variable CAR is the same as the pointer in J. Temporary entities can be destroyed one at a time, and their storage returned to free storage.

Sets are logical groupings of entities, and can be used to identify interrelationships such as queues. In the simple model outlined here, each pump (assuming the layout warranted it) would OWN (have) its own queue. Although the queue may typically have no occupants, the data structure is set up so that a potential queue exists for each pump. The potential members of each queue are CAR's. Defining every GAS.PUMP as owning a queue automatically provides three extra attributes for every pump created, called F.QUEUE, L.QUEUE and N.QUEUE. These are respectively; a pointer to the first CAR in the queue, a pointer to the last CAR in the queue, and the number of cars in the queue. Defining CAR as possibly belonging to a queue automatically provides three extra attributes for each CAR created, called P.QUEUE, S.QUEUE, and M.QUEUE. These are respectively; a pointer to the car's predecessor in the queue, a pointer to its successor in the queue, and a flag marking whether or not it is (1) or is not (0) a member of the queue. These set ownership and membership attributes are the mechanisms by which sets are defined. They link owners and members together in structured lists, expressing <u>all</u> the information concerning sets which is maintained by the system. For example, if the gas station has three GAS.PUMPs, numbered 1, 2, and 3, and the number of cars awaiting service at each pump is 0, 1, and 3 respectively, then the data structure generated will be as follows:

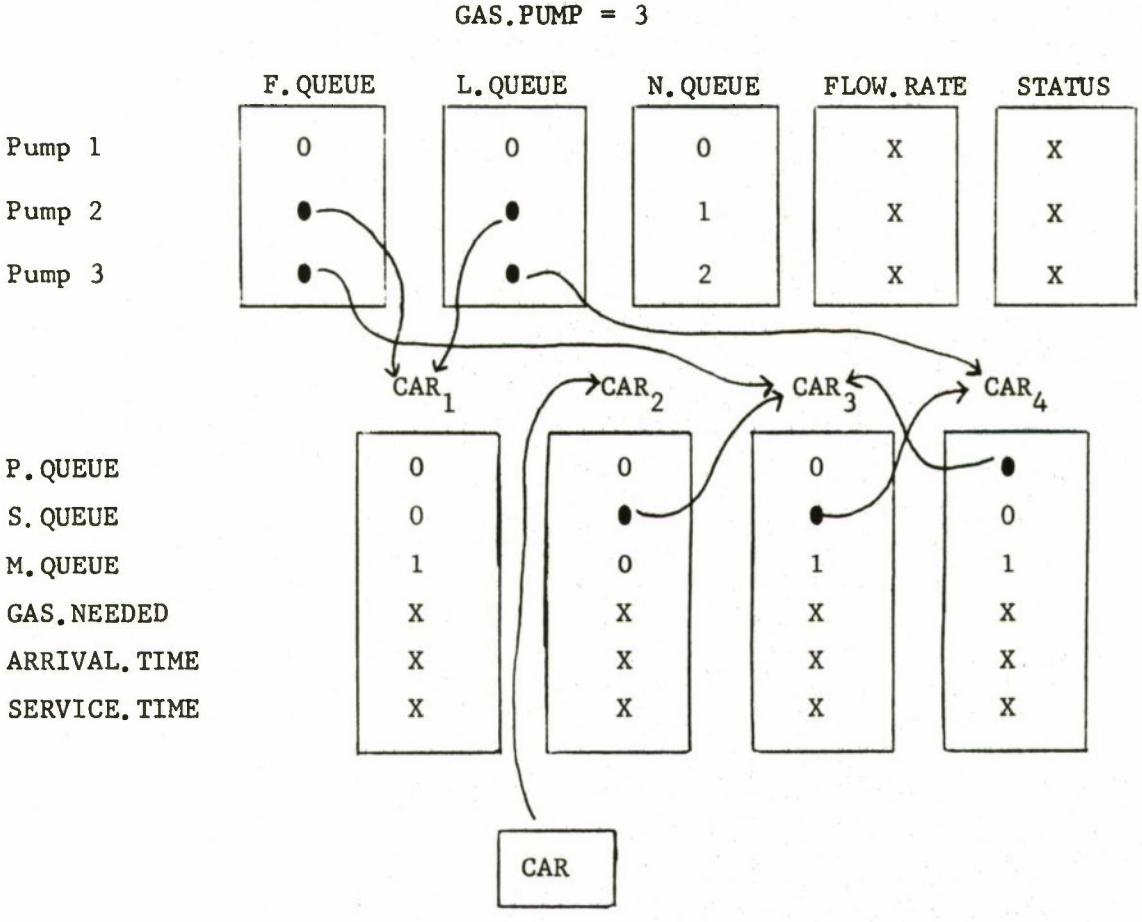


For illustrative purposes, the global variables GAS.PUMP and CAR are arbitrarily assumed to point to the last of each type of entity defined above.

When servicing of a car is completed, the car next in line for servicing can be accessed simply by "REMOVE THE FIRST CAR FROM QUEUE". This statement removes the first car (CAR_2) from the queue currently identified by the implicit reference QUEUE(GAS.PUMP) and assigns the pointer to this car to the global variable CAR. The references GAS.NEEDED, GAS.NEEDED(CAR) and GAS.NEEDED(CAR_2) are then equivalent, and the first is usually preferred since it can be used to reference the GAS.NEEDED by other cars when CAR points to them. It is therefore short, unambiguous, and implicit.

SYSTEM DYNAMICS

Changes in state of a system are represented by altered numerical value(s) of one or more attributes of one or more entities of the system. Thus the removal of CAR_2 from the QUEUE before the GAS.PUMP of the previous illustration results in the following data structure:



Changes in system state, which are accomplished through program statements that alter attributes, are typically in event routines supplied by the programmer.

In SIMSCRIPT's world view, the basic unit of action is called an activity. The two important aspects of activities are (1) that

they take time, and (2) that they (potentially) change the state of the system. Most activities are bounded by two "events," a start activity event and a stop activity event. Events take zero simulated time and produce changes in data structures which reflect the change in system state occurring at that instant of time. The passage of time which occurs during an activity is represented as a time delay factor between start and stop events.

Continuing the previous example, when a CAR has had its servicing completed, it will leave the gas station. This action is represented by the destruction of the particular temporary entity which represents the car (some statistical gathering may be performed first). The STATUS attributes of the GAS.PUMP and the ATTENDANT servicing the car would be set to zero to indicate that they are now idle. A test would then be made to see if the QUEUE (GAS.PUMP) had any members. If so, conditions are established which result in the initiation of servicing for the next car. This is done by "scheduling" an event to accomplish these actions at the current time. Typically, the event which starts an activity also schedules the completion of the activity at some future point in time.

The occurrence of events in their proper temporal sequence is accomplished by the SIMSCRIPT system. The mechanisms employed are an artificial system clock and a timing routine. The changes in system state represented by an event are accomplished by an event routine which is called by and which returns to the timing routine. The timing routine maintains a set of all events which are scheduled to occur, and it calls event routines in their proper order, updating the clock in between as required. Thus a SIMSCRIPT simulation program usually contains a small main program consisting primarily of initialization statements which, in addition to setting up data structures, must schedule at least one event before a "START SIMULATION" statement is executed. This statement transfers control to the timing routine, which examines its set of scheduled events to find the earliest, advances the system clock to that time, and transfers to that event routine. If the timing routine runs out of events, it returns control to the main program at the statement following START SIMULATION.

This methodology is implemented through using the data structures defined earlier. "Event notices" are actually temporary entities created every time an event is scheduled. Event notices are filed in a timing set, which is a set organized by event type and then by scheduled time of occurrence. If more than one event can occur at the same instant of time, the programmer can specify a priority ordering between different event types, as well as break ties within event types (e.g. servicing of two cars is scheduled to

be completed at the same instant) by high or low values of any attribute(s). Unless the programmer specifies otherwise, the system destroys each event notice before passing control to the proper event routine.

SECTION IV

LANGUAGE DIFFERENCES BETWEEN SIMSCRIPT I AND SIMSCRIPT II

As outlined in the previous section, there are essentially two distinct SIMSCRIPT languages, I - I.5 and II - II Plus. I.5 and II Plus are derivatives of their predecessors, constituting different implementations rather than different languages. At present, SIMSCRIPT I is relatively obsolete, and the present implementations of neither II nor II Plus permit the full range of statements identified for the SIMSCRIPT II language. The most useful means of defining the features, differences, and similarities between these various versions of SIMSCRIPT is probably to contrast the languages first, and then to characterize those aspects which derive from the implementations. The latter subject is treated in the next section. As outlined earlier, all versions of SIMSCRIPT provide the same data structures and event mechanisms. The primary language differences result from the techniques by which these features are specified and from augmented general programming power provided by II Plus.

DATA STRUCTURE DEFINITION

In SIMSCRIPT I, all data structures are specified on a definition form in a strict format which requires precise card column spacings for the identification of temporary entities, permanent entities, event notices, all attributes, and all set memberships. Various entries on this form are left or right justified, according to the data type. Variable names of any type are limited to five alphanumeric characters. Another form is provided for initialization of attribute values, and the meanings of the entries on this form are difficult to decipher because the column spacings are exact and its entries are almost exclusively numeric.

In contrast, SIMSCRIPT II accomplishes the same specification of data structures through free form English-like statements. First, variable names are not limited as to length, and may include periods, so that a name like PART.NUMBER is valid for any data item. Data structures are identified through a section of the program called a PREAMBLE, which must precede all other sections. Statements in this section are all definitional, i.e. non-executable, and serve to identify variables which are "global" to a simulation program. Global variables, like COMMON in FORTRAN, are accessible to all routines present in a program (variables declared on the definition form of SIMSCRIPT I are global also). Entities, attributes, and

sets are specified through EVERY statements, which signal the compiler that the entities named possess the structure defined. In terms of the example of Section III, one could write:

```
PREAMBLE

TEMPORARY ENTITIES

EVERY CAR HAS A GAS.NEEDED, AN ARRIVAL.TIME, AND A SERVICE.TIME,
     AND MAY BELONG TO A QUEUE

PERMANENT ENTITIES

EVERY GAS.PUMP HAS A FLOW.RATE AND A STATUS AND OWNS A QUEUE

EVERY ATTENDANT HAS A DELAY AND A STATUS
```

The name following EVERY (e.g. CAR) is identified as an entity of the type specified previously (e.g. TEMPORARY). The attributes of entities are denoted in a name list following HAS. Possible set memberships are indicated following BELONG(S) TO, and set ownership(s) of the name(s) following OWNS. The set ownership and membership attributes are automatically generated for each entity created. Unlike the mechanisms employed in SIMSCRIPT I, the above method of data definition is highly readable, helping significantly to produce self-documenting programs. The clarity of the structure of the model provided in SIMSCRIPT II aids significantly when that structure is to be altered, as is often the case in simulation studies. One author[7] "experienced that about 75 percent of the debugging effort (with SIMSCRIPT I.5) is spent correcting Definition and Initialization Forms". The free-form English-like syntax of the SIMSCRIPT II PREAMBLE which replaces these forms facilitates debugging through making the relationships specified obvious even to the casual reader. Some people at the Aerospace Corporation have used SIMSCRIPT II PREAMBLE statements for designing systems and communicating the designs, though never intending to simulate the systems thus described.

EXECUTION TIME FACILITIES

There are two execution-time aids to debugging in SIMSCRIPT II which also warrant mention. A global variable called BETWEEN.V is defined by the system. Its contents, which are initialized to zero,

are tested just before the timing routine transfers to any event routine. If the programmer has altered BETWEEN.V, then the system will transfer to the routine named. For example, the statement LET BETWEEN.V = 'TRACE' sets the address of the routine TRACE in the contents of BETWEEN.V. Before each event is executed, the programmer can perform whatever diagnosis he desires or collect statistics which reveal the dynamic properties of his model. The steps taken at this point are determined by the programmer in the routine TRACE (or any other named routine) which he writes.

The second powerful debugging aid is achieved through defining variables as "monitored." A monitored variable has associated with it both a storage location and a program. It therefore represents a new data type, since it has the features of both a function and a variable. To use the monitoring feature, one must explicitly declare a variable as monitored in a DEFINE statement, e.g.,

(a) DEFINE X AS AN INTEGER VARIABLE MONITORED ON THE RIGHT

(b) DEFINE Y AS A REAL, 2-DIMENSIONAL ARRAY MONITORED ON LEFT AND RIGHT

One must also define right and/or left handed functions designed to perform the right and/or left monitoring.

Functions, e.g. SIN(A), are usually employed to compute a value from one or more arguments, and can be treated in expressions as though they were a variable, e.g., C**2-5*(SIN(A)**3+COS(B)). These are "right-handed" functions which appear to the right of an equals sign and are used to compute values. Left-handed functions, on the other hand, receive values. In SIMSCRIPT, it is legal to say LET FUNCTION.NAME (I) = A, but one must then define a LEFT ROUTINE FUNCTION.NAME GIVEN I. This routine must include as its first executable statement ENTER WITH X, which takes the value of A and assigns it to X. From this point on, the routine can perform any legal SIMSCRIPT operations. The utility of this construct becomes apparent when it is used in monitoring variables.

Suppose a programmer suspects that some problems with his program result from references to a particular array. He can then specify, as in (b) above, that his array Y is monitored on both left and right and provide routines defined that way. This is the only change made to his program. However, he now has the ability to check every retrieval from storage (get) through a right hand function

LET Z = Y(I,J)

and every assignment to storage (put) through a left hand function

```
LET Y(I,J) = 3*A**2.
```

Monitoring can be used for checking for valid subscripts, editing data during reading, transformation of data for printing, etc. -- whatever purposes one might wish to achieve every time a storage retrieval or assignment is made. The biggest benefit with this feature is not that it can be done -- one can always insert a statement before every put or get which transfers to a subroutine -- but that it is done automatically, without cluttering up a program with odd-looking statements. Thus if in debugging it becomes worthwhile to monitor a variable, the additions to a program which accomplish this are minimal and appear in a few well-delineated locations in a program. When the bug has been discovered and eliminated, the monitoring program statements to be removed can be located easily. In contrast, providing for monitoring a variable through direct coding requires much more work and introduces its own opportunities for error through overlooking program locations that either require a transfer when debugging or require the removal of a transfer when debugging is completed.

LANGUAGE ADVANTAGES OF SIMSCRIPT II

Many of the improvements incorporated into SIMSCRIPT II represent changes in more than syntax or semantics from its predecessor. A partial survey of these new features is provided below.

1. Dynamic Storage Allocation.

All arrays are dimensioned at execute time in SIMSCRIPT II. In SIMSCRIPT I this is true except for local arrays (i.e. arrays which are not system variables, but declared in a subroutine), which must be dimensioned as in FORTRAN.

2. Releasable Programs.

All routines (subroutines and functions) may be declared as RELEASABLE. In large programs this feature permits discarding a routine if it is no longer required so that its memory space can be used for other purposes. The statements which initialize a simulation model, for example, are performed only once in any run unless the model is restarted. These steps, which often occupy a significant amount of memory, can be isolated in an initialization routine and RELEASED after they have been performed to provide extra memory during simulation.

3. Dynamic Program Relocation.

Dynamic program relocation, i.e. not only releasing routines but later restoring them in memory in operable form, is defined in SIMSCRIPT II through LOAD and SAVE statements. The provision of this facility greatly enlarges the effective memory capacity available for program storage.

4. Recursion.

All routines are recursive in SIMSCRIPT II, unless declared otherwise. Hence the following routine, when called once by a program, will return N factorial to that program.

```
ROUTINE FOR FACTORIAL (N)

IF N = 1, RETURN WITH 1

OTHERWISE RETURN WITH FACTORIAL (N-1)*N

END
```

5. Free-Field Programs.

SIMSCRIPT II statements may be punched in any card columns, and more than one per card is permissible. The only rules which limit the concept of a continuous program string are imposed to simplify the punching of comments and to retain visual integrity of variables, e.g. variable names cannot be split between cards. These features permit indenting statements to produce a more readable program and eliminate errors due to off-column punching. The only word which cannot be used as the name for a variable or label in a SIMSCRIPT II program is AND.

6. Input/Output

The input of data can proceed either under formatted control or under a free-form specification by which blank characters mark the separation between input fields. Thus it is possible to say

```
READ X, Y, Z
```

without any FORMAT statement. Enough data cards will be read to locate three values. A similar capability exists for output, e.g.

```
PRINT 1 LINE WITH X,Y,Z LIKE THIS

X = ****.*     Y = **     Z = **
```

The free-form input capability listed above is the mechanism which permits free-field SIMSCRIPT II programs, since the compiler is written in SIMSCRIPT II. In contrast, SIMSCRIPT I I/O is almost identical to FORTRAN, although a report generator (RPG) capability is provided. SIMSCRIPT II offers no RPG facility, only page-heading statements. Output is, however, simply and straightforwardly achieved.

7. Mode and Dimensionality Specification.

Unlike FORTRAN and SIMSCRIPT I, variables beginning with I, J, K, L, M, and N may be real numbers or integers in SIMSCRIPT II. The mode and dimensionality of all variables is set implicitly if not declared explicitly. The compilation process initiates with background conditions which specify variable mode as real (rather than integer), and dimensionality as zero (rather than 1, or 2, or . . .). These background conditions are overridden by explicit declaration (DEFINE IJK TO BE AN INTEGER, 1-DIMENSIONAL ARRAY) and changed by specification (NORMALLY, MODE IS INTEGER, DIMENSION IS 2).

8. User Access to System.

The programmer has complete access to attributes, constants, entities, functions, routines, sets, and variables defined by the SIMSCRIPT II system. Not all of these are accessible in SIMSCRIPT I.

9. Storage References.

In SIMSCRIPT I, a different subroutine for storage retrieval (get) and assignment (put) is generated for every global variable. Hence there are many subroutines and calls to subroutines. SIMSCRIPT II generates code that can make these references directly through a PREAMBLE generated control section called PRMB.

10. Attribute Storage

For each group of eight words required to store SIMSCRIPT I attributes a different block of core storage is utilized. Hence storage referencing for entities with many attributes becomes indirect and inefficient relative to SIMSCRIPT II.

11. Event Scheduling Priority.

SIMSCRIPT I has no provision for declaring priority of occurrence between events. SIMSCRIPT II has the PRIORITY (different event types) and BREAK TIES (by attribute(s) for the same event) statements to accomplish this function. Priorities among different event types are implicitly established in SIMSCRIPT I by the order of event appearance on the Definition Form. Thus a PRIORITY is provided by default; the fact that it is not explicitly declared does not matter as long as the implicit ordering is correct. Significant redesign of the Definition and Initialization Form entries may be required to alter priorities, however.

12. OLD PREAMBLE Feature.

Prefacing PREAMBLE by OLD inhibits the production of set filing and removing routines, the timing routine, and LIST routines, and speeds the compilation process. A parallel capability is available in SIMSCRIPT I, but on a card-by-card basis for each definition card. This is useful when recompiling due to program changes that do not affect global variables.

13. DEFINE Word TO MEAN Words.

This statement can be employed as a shorthand, e.g.

DEFINE LOCAL TO MEAN DEFINE I, J, K, L, M, and N

AS INTEGER VARIABLES

With this statement in the PREAMBLE of a program, the single word, LOCAL, in each subroutine defines I, J, K, L, M, and N as integers. No similar statement is provided in SIMSCRIPT I.

SECTION V

IMPLEMENTATION DIFFERENCES - SIMSCRIPT I, I.5, II, II PLUS, AND II.5

The five versions of SIMSCRIPT can be contrasted in two ways: through detailed variations in language provisions and by differences in more global measures such as execution speed and coverage of diagnostics. These subjects are treated below in the order mentioned. The standards for language comparisons are the language specifications for versions I and II discussed in the previous section.

LANGUAGE PROVISIONS

Although the language differences between SIMSCRIPT I and I.5 are minor, the latter does provide for:(8)

1. Local variables in excess of five characters, e.g. THETOTALINVENTORY.

2. Symbolic labels anywhere in columns 2 - 5. In SIMSCRIPT I only right adjusted integers were valid.

3. Labeled blank statements so that labels can be made equivalent.

4. Elimination of two problems arising from the FORTRAN integer convention.

5. Several other alterations which facilitate input-output and ease syntax restrictions.

The RAND implementation of SIMSCRIPT II is consistent with the language texts issued by both RAND and Prentice-Hall, but a number of the statement types specified in the design of the language are missing. These statements yet to be implemented are outlined below.

1. CLOSE, ADVANCE, and BACKSPACE

These commands affect the positioning of data sets. The end-of-file marker set by a CLOSE statement can also be accomplished

through REWIND. ADVANCE and BACKSPACE move the specified I/O device forward or backward a specified number of files and can be accomplished through assembly code.

2. Column Repetition

This feature, provided in SIMSCRIPT I, permits large arrays of data to be printed with the column indices (e.g. 1 to 50 on page 1 and 51 to 100 on page 2) handled automatically.

3. LIST ATTRIBUTES OF EACH Entity

Several forms of this statement result in the listing, in a predefined format, of one or more attributes of one or more entities. It is useful in debugging, tracing dynamic properties, etc.

4. Automatic Entity Checking When Entity Destroyed

An execution error should be flagged when an entity is destroyed that is still a member of a set. Since the RAND SIMSCRIPT II system does not recognize this error, it could lead to errors that are difficult to debug.

5. Left-Handed Functions

(Described previously.)

6. Monitored Variables and Attributes

(Described previously.)

7. All TEXT Features

A number of commands are specified which permit the definition, manipulation, input, output, and mode conversions of character strings, none of which are presently available.

8. BREAK TIES

(Described previously.)

9. Event Arguments in SCHEDULE and EVENT Statements

The language specification permits the syntax which follows: SCHEDULE AN event GIVEN expression list AT time expression. This statement creates an event, sequentially assigns the attributes

listed in the expression list, and files the event in the events set ordered by its time of occurrence (time expression). The inability to assign attributes in a SCHEDULE statement requires only the addition of an assignment statement. If a BREAK TIES ordering is specified, however, three statements (CREATE, LET attributes = values, and FILE) are required, since the SCHEDULE statement does the filing automatically with attribute values all equal to zero. Thus the BREAK TIES ordering can only be maintained by assigning attribute values before filing in the events set.

10. BEFORE and AFTER

These statements, which appear in a program PREAMBLE, are helpful in debugging. One can use them to direct the calling of various debugging routines before or after the performance of certain specified operations. Thus, BEFORE or AFTER CREATING or DESTROYING an entity, SCHEDULING or CANCELING an event, or FILING in or REMOVING from a set, a specified routine can be called. The arguments which are transmitted to the operation being monitored (e.g. CREATE) are automatically transmitted to the monitoring routine.

11. ACCUMULATE, TALLY, DUMMY, and RESET

ACCUMULATE and TALLY are statements which appear in the PREAMBLE of a program which automatically generate a powerful statistics-gathering capability for each of the variables mentioned in the statement. One can write:

ACCUMULATE AVG.QUEUE AS THE MEAN AND MAX.QUEUE

AS THE MAXIMUM OF N.QUEUE

Every time N.QUEUE changes, the appropriate accumulations are made so that the variables AVG.QUEUE and MAX.QUEUE respectively contain the average number of members and the maximum number of members in QUEUE. This facility permits operating (i.e. non-PREAMBLE) portions of programs to be kept free of data collection and data reduction statements. TALLY treats every observation value with equal weight, e.g. the SUM of X $=\sum X_i$, while ACCUMULATE produces a time-weighted sum, e.g. the SUM of X $=\sum X_i \Delta t_i$, where Δt_i represents the time duration of X_i. Thus the value of AVG.QUEUE above would be the time average length of the QUEUE. DUMMY simply reduces the storage allocation used for these purposes when possible. RESET restores the statistical counters employed to zero.

12. RANDOM Variables

This provision permits the random sampling from an arbitrary distribution defined by the programmer. This is accomplished through storing a table containing a description of the cumulative distribution function. The concept and effect are equivalent to FUNCTION blocks with random number seeds in GPSS. Variables may be defined as step (discontinuous) or linear (continuous) functions, as in GPSS.

13. ORIGIN.R

This is a routine which permits the specification of time in a calendar format (e.g. 3/24/71 09 45 represents 9:45 in the morning of March 24, 1971) through establishing a time origin against which such calendar specifications can be matched.

SIMSCRIPT II Plus does not yet include the entire repertoire of language statements defined for the SIMSCRIPT II language. The following statements, which have not been implemented, have been discussed previously.

1. CLOSE, ADVANCE, BACKSPACE

2. Column Repetition

3. SAVE, LOAD

4. TEXT features

5. ACCUMULATE, TALLY, DUMMY, and RESET

6. ORIGIN.R routine

Simulation Associates had planned to implement some of these statements this year, but C.A.C.I.'s time schedule may be quite different.

Several additions and alterations to the defined SIMSCRIPT II language have been incorporated in II Plus. These are:

1. Routines no longer have to be declared as RELEASABLE. The programmer can simply write RELEASE routine name.

2. The standard TRACE output routine called by the run-time error monitor now calls a routine named SNAP.R. SNAP.R is present as a null routine in the run-time library, i.e. as ROUTINE SNAP.R RETURN END, but may be replaced by the user's own SNAP.R routine. Hence, important data items can be printed whenever execution errors are encountered.

In its announcement of SIMSCRIPT II.5, C.A.C.I. has thus far identified only one significant change from II Plus, the provision of double-precision floating-point arithmetic. Evidently, in some simulation applications a loss of numeric significance has resulted from the use of single-precision floating-point. This addition is currently being implemented, and others are expected to follow.

NON-LANGUAGE FEATURES

Other important differences between the various versions of SIMSCRIPT derive from features other than the language available to the programmer. Among these are diagnostics for error correction, typical core storage requirements, and speed in compilation and execution.

1. Diagnostics

The two recent SIMSCRIPT implementations, II and II Plus, provide a fairly extensive set of diagnostics during both compilation and execution. In contrast, I.5's diagnostics are rather minimal, as can be seen below.

Table I

Number of Error Diagnostics

SIMSCRIPT I.5, II, and II Plus

SIMSCRIPT Version	Number of Messages Provided During Compilation	Execution
I.5	29	0
II	79	105
II Plus	86	107

Another feature of SIMSCRIPT II and II Plus is that the compiler "corrects" as many errors as it can and ignores statements containing the remainder. Thus, when a significant number of programmer syntax errors can be corrected appropriately, a run is not lost. This correct or ignore feature is used to force the execution of

every program except those with syntax error(s) in the PREAMBLE section that are likely to invalidate all subsequent routines. The philosophy behind this is that valuable information is gained through executing as far as possible. In contrast, I and I.5 stop upon encountering an error in the Initial Conditions Data Deck. It may take several runs to discover all errors present in this deck.

2. Core Storage Requirements

Both II and II Plus normally require 150K bytes to compile using a compiler overlay, 180K bytes without the overlay, and 52K bytes to execute simulation models. Although comparable figures for I.5 are not available, SIMSCRIPT I was implemented in a 32K word machine (36 bits/word). Promotional material distributed by Simulation Associates claims that II Plus generates smaller programs making better use of available core than SIMSCRIPT I or I.5.

3. Compilation and Execution Speed

There is little doubt that the II Plus implementation compiles programs faster than SIMSCRIPT II. The range of estimates available suggest that II Plus compilation takes roughly half as much time, due largely to the fact that the RAND version uses IBM's assembler, and the II Plus version utilizes a subset of the IBM compiler written by Simulation Associates. Robert Parente[6] suggests that completing a large simulation study would cost three times as much with the SHARE version as with II Plus. At present, rough estimates for II Plus[7] are compilation speed of 500-1000 cards per minute (depending upon statement mix) and execution speed of about one millisecond per statement. For comparison, GPSS executes about 1 block per millisecond. The clients of Simulation Associates have expressed more concern with compilation speed than execution speed.

Simulation Associates distributed a performance comparison between SIMSCRIPT I.5 version 1.0 and SIMSCRIPT II Plus Release 2A. Their data are displayed in Table II, which shows that II Plus is somewhat more demanding of space and time during compilation than I.5, but very much more efficient during assembly. The II Plus data and Mr. Parente's estimate are inconsistent, but recent changes to the compiler have improved its speed, so that the Table 2 figures may be slightly out of date.

---

[6] Formerly of Simulation Associates.

[7] Given by Robert Parente.

Table II

SIMSCRIPT I.5 and II Plus Performance Comparisons

Job Shop Simulation Model

(360/65 Timings)

Characteristic	SIMSCRIPT I.5	SIMSCRIPT II Plus	
Source Statements	115	105	
Data Cards	63	9	
CPU Seconds			
- Compilation	16.2	13.4	18.2*
- Assembly	14.4	2.5	3.6*
- Execution	.6	.4	
- Total	31.2	16.3	22.2*
CPU Milliseconds/Source Statement			
- Compilation	140.9	127.6	173.3*
- Assembly	125.2	23.8	34.3*
- Execution	5.2	3.8	
- Total	271.3	155.2	211.4*
Core Required (Thousands of Bytes)			
- Compilation	108	146	
- Assembly	104	60	
- Execution	84	42	

*Figures are for initial compilation of entire program. Preceding figure is for compilation of all programs using OLD PREAMBLE feature.

One reason why II Plus execution is faster than I.5 stems from its efficiency in dynamic storage allocation. With I.5, garbage collection was nearly continuous. II Plus keeps lists (actually SIMSCRIPT sets) of identical segments of free core. For example, the temporary entity CAR and the event notice ARRIVAL may each require 6 words of core. When one of these is destroyed or cancelled, its former record (slot in core) is put into the set of all 6-word records that have been returned to free storage and that are currently unused. When one of these 6-word records is created, the procedure is:

(1) Check the 6-word set to see if any available. If so, take the first, if not,

(2) Go to the next larger list and try there. If unsuccessful,

(3) Use GET MAIN of O/S 360. If unsuccessful,

(4) Try garbage collection until enough space is provided. If impossible,

(5) Flag an error.

An extension of SIMSCRIPT II, called the Extendable Computer System Simulator (ECSS) has been developed at RAND. It permits simulations with a process orientation, and contains SIMSCRIPT II as a language subset. Although it is still in a field test status, when operational it could provide a powerful tool for computer systems simulation. Thus its existence is a cogent argument in favor of SIMSCRIPT II or II.5 rather than I.5.

SECTION VI

CONCLUSIONS

The data structures and event mechanisms provided in SIMSCRIPT, together with the flexibility and power of the language, permit the modeling of complex systems with both brevity and clarity. Reviews[8] of simulation languages generally consider SIMSCRIPT to be one of the most powerful of the simulation languages currently available. This is only one of the criteria by which a language should be selected, however. Others are listed below.

CRITERIA FOR LANGUAGE SELECTION

The reasons cited for selecting one computer language instead of another include:

(a) Programming concepts,

(b) Usability,

(c) Readability,

(d) Training required,

(e) Cost,

(f) Flexibility,

(g) Documentation,

(h) Support,

(i) Modeling concepts and

(j) Transferability

SIMSCRIPT, particularly II Plus, provides extensive language level advantages, well-designed programming concepts, modeling constructs and programming flexibility that permit natural system descriptions and highly readable, self-documenting programs, and reasonable support and documentation. On the negative side, when compared to more

[8] See References 9-15.

structured languages like GPSS, SIMSCRIPT is more difficult to learn and debug, and takes longer to code. If many replications of a particular simulation experiment are to be done for statistical validity, and if many experiments are involved, then the more efficient code generated by SIMSCRIPT will be preferable to the interpretive operation of GPSS. However, if the rapid production of a small number of runs is emphasized, then GPSS should be used.

Although the reasons cited can and do influence language selection, the actual decision process on a case-by-case basis often reduces to a consideration of (1) the programmer's capability and (2) availability of the system at your installation. Language selection decisions therefore frequently produce non-optimal overall results. Given a limited simulation task today, one would doubtless select GPSS (if available at the installation and known to the programmer). Every time in the future that the same situation arises, the decision would be the same; but if a "joint" decision considering all of this work could be made, the language selected might well be other than GPSS, i.e. an investment in language availability and programmer knowledge might be warranted. There is no escape from imperfect foresight, but one may as well make best-guess projections of future requirements and incorporate these projections into the decision-making process. A review of past and projected future simulation efforts may prove worthwhile from this standpoint.

VERSIONS OF SIMSCRIPT

Three versions of SIMSCRIPT are presently candidates for utilization: I.5, II, and II Plus-II.5. Although II is free, it is not supported, is significantly slower than II Plus, and only recently has become operative under MFT. An installation might economically use SIMSCRIPT II for limited applications, or as a test to determine programmer acceptance and utilization. Once utilization of SIMSCRIPT II passes a certain threshold, however, it will be less expensive to buy II Plus.

Although the present status of II Plus-II.5 is somewhat evolutionary, it already offers many advantages over I.5. Its only disadvantage is cost, although cost comparisons are muddied by a dissimilarity in quoted price terms. C.A.C.I. offers I.5 on the following bases: (a) two year lease for $12,400, including system maintenance and updating, with subsequent years on a yearly basis for $3,000, (b) perpetual usage for $15,000, including two years of system maintenance and updating, with subsequent years for $1,500, (c) one year lease for $7,200, including system maintenance and

updating, with a second year at $7,200 and the third and succeeding years for $3,000. C.A.C.I. now offers II.5 for a flat $500/month, although optional pricing arrangements may be offered in the future. For a five year period, the minimum I.5 cost would therefore be $19,500, while II.5 would cost $30,000. If the simulation efforts involved are large and time-consuming, the faster operation of II.5 may well result in its being the most economical. It also offers the many programmer advantages detailed earlier. In the opinion of John Maguire, who as Senior Vice President and Director of Technical Operations for C.A.C.I. speaks as the vendor of both versions, although I.5 is presently used in far more installations than II or II.5, and although II.5 is still in a state of flux as additional statements are implemented and the compiler is speeded up, in five years more people will be using II.5 than I.5.

SIMULATION OF COMPUTER SYSTEMS

Developing a valid model of a computer system requires: (1) expertise in the techniques of simulation, including knowledge of both statistics and the computer simulation language employed, (2) appreciation for the inner workings of system software and its points of interaction with applications programs and service routines, (3) knowledge of hardware architecture, interactions, and timings, and (4) sufficient data concerning applications programs to generate valid workload models. The effort required is large, and the talents demanded diverse. The specification, design, coding, debugging, statistics gathering and reduction, and finally the validation of the simulation model cannot be accomplished in short periods of time. For these reasons, simulation is a questionable tool for evaluating vendor proposals tendered in a normal procurement cycle, regardless of the language used.

Simulation can be employed, however, when the time and resources are available, as is often the case during system design. Examples of this are readily available - - the GPSS simulation of the Advanced Airborne Command Post and many computer manufacturers' simulations of hardware and/or operating systems. The utility of SIMSCRIPT to ESD would appear to be in future design and feasibility studies in which the time and resources are available and in the many subsidiary problems which arise that do not require the modeling of a complete computer system.

The decision to provide SIMSCRIPT as a simulation tool at MITRE/ESD hinges primarily upon expectations concerning the size and character of future simulation efforts. SIMSCRIPT requires more investment than many of its competitors (particularly GPSS) but is

capable of producing better results. To do so requires a higher level of programmer expertise, at least for simple models. The coding of complex models in GPSS can become quite involved if the language (block) constructs do not identify easily with system elements.

REFERENCES

1. Knuth, D. E., The Art of Computer Programming, Volume 1, Addison-Wesley, Reading, Mass., 1968., p. x.

2. DesRoches, J. C., "Survey of Simulation Languages and Programs," The MITRE Corporation, ESD-TR-71-227 (MTR-2040), January 1971.

3. Kiviat, P. J., R. Villanueva and H. M. Markowitz, The SIMSCRIPT II Programming Language, The RAND Corporation, R-460-PR, October 1968., p. v.

4. Kiviat, P. J., R. Villanueva, and H. M. Markowitz, The SIMSCRIPT II Programming Language, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1968.

5. Kiviat, P. J. and R. Villanueva, "The SIMSCRIPT II Programming Language: Reference Manual," The RAND Corporation, RM-5776-PR, October, 1968.

6. Kiviat, P. J., H. J. Shukiar, J. B. Urman, and R. Villanueva, "The SIMSCRIPT II Programming Language: IBM 360 Implementation," The RAND Corporation, RM-5777-PR, 1969., p. 41.

7. Wyman, F. P., Simulation Modeling: A Guide to Using SIMSCRIPT, John Wiley & Sons, Inc., New York, 1970., p. 202.

8. Consolidated Analysis Centers, Inc., "SIMSCRIPT I.5," Santa Monica, California, July 1967., p. 4-16.

9. Dahl, O. J., "Discrete Event Simulation Languages," Lectures Delivered at the NATO Summer School, Villard-de-Lans, September 1966.

10. Krasnow, H. S., "Dynamic Representation in Discrete Interaction Simulation Languages," Digital Simulation in Operational Research, S. H. Hollingdale, Ed., Lectures Presented at the NATO Scientific Affairs Conference, Hamburg, Germany, September 1965, p. 77-92.

11. Kiviat, P. J., "Digital Computer Simulation: Computer Programming Languages," The RAND Corporation, RM-5883-PR, Santa Monica, California, January 1969.

12. Krasnow, H. S. and R. A. Merikallio, "The Past, Present, and Future of Simulation Languages," Management Science, November 1964, p. 236-267.

REFERENCES (Concluded)

13. Freeman, D. E., "Programming Languages Ease Digital Simulation," Control Engineering, November 1964, p. 103-106A.

14. Teichroew, D. and J. F. Lubin, "Computer Simulation - Discussion of the Technique and Comparison of Languages," Simulation, Volume 9, No. 4, October 1967, p. 181-190.

15. Tocher, K. D. "Review of Simulation Languages," Operations Research Quarterly, Volume 16, No. 2, p. 189-218.

BIBLIOGRAPHY

Boehm, B. W., "Computer Systems Analysis Methodology: Studies in Measuring, Evaluating, and Simulating Computer Systems," The RAND Corporation, R-520-NASA, 1970.

Control Data Corporation, Control Data 6400/6500/6600 Computer Systems SIMSCRIPT Reference Manual, Pub. No. 60178300, Palo Alto, California, 1968.

Gainen, L., "Complex Business Problems? Try SIMSCRIPT, a Powerful Simulation Language," Computer Decisions, April 1970, p. 52-56.

Geisler, M. A. and H. M Markowitz, "A Brief Review of SIMSCRIPT as A Simulating Technique," The RAND Corporation, RM-3778-PR, 1963.

Gordon, G., Systems Simulation, Prentice Hall, Inc., Englewood Cliffs, N. J., p. 239-275.

Karr, H. W., H. Kleine and H. M. Markowitz, "SIMSCRIPT I.5," California Analysis Center, Inc., Santa Monica, California, 1966.

Kiviat, P. J., "Development of Discrete Digital Simulation Languages," Simulation, February, 1967, p. 65-70.

Kiviat, P. J., "Introduction to the SIMSCRIPT II Programming Language," Digest of the Second Conference on Applications of Simulation, December 2-4, 1968, New York City.

Kiviat, P. J., "Simulation Programming Using SIMSCRIPT II," The RAND Corporation, P-3861, 1968.

Kiviat, P. J., "The SIMSCRIPT II Programming Language," SHARE Contributed Program Library Submission 360D-03.2.014, 1969.

Kosy, D. W., "Experience with the Extendable Computer System Simulator," The RAND Corporation, R-560-NASA/PR, 1970.

Kosy, D. W., "Experience with the Extendable Computer System Simulator," Third Conference on Applications of Simulation, December 8-10, 1969, Los Angeles, p. 235-243.

BIBLIOGRAPHY (Concluded)

Kosy, D. W., "The Extendable Computer System Simulator Language Specification Manual," The RAND Corporation, R-561.

Markowitz, H. M., B. Hausner, and H. W. Karr, SIMSCRIPT - A Simulation Programming Language, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1963.

Markowitz, H. M., "Simulating with SIMSCRIPT," Management Science, Vol. XII, No. 10, June 1966, p. 396-409.

Neilsen, N. R., "ECSS: An Extendable Computer System Simulator," The RAND Corporation, RM-6132-NASA, 1970.

Neilsen, N. R., "ECSS: An Extendable Computer System Simulator," Third Conference on Applications of Simulation, December 8-10, 1969, Los Angeles, p. 114-129.

Neisius, W. V., E. D. Katz, and D. B. Townsend, "Technical Note for SIMSCRIPT Users," Digest of the Second Conference on Applications of Simulation, December 2-4, 1968, SHARE/ACM/IEEE/SCi, p. 45-47.

Simulation Associates, "Job Shop Simulation Model," undated release, Los Angeles and White Plains.

Simulation Associates, Inc., "SIMSCRIPT II Plus User's Manual," Los Angeles and White Plains, 1970.

Southern Simulation Service, Inc., "SIMSCRIPT I.5," undated memorandum.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) The MITRE Corporation P. O. Box 208 Bedford, Massachusetts 01730	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

A GUIDE TO THE POTENTIAL USE OF SIMSCRIPT

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (First name, middle initial, last name)

Peter R. Burleson

6. REPORT DATE SEPTEMBER 1971	7a. TOTAL NO. OF PAGES 41	7b. NO. OF REFS 15
8a. CONTRACT OR GRANT NO. F19(628)-71-C-0002 b. PROJECT NO. 5720 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-71-346 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) MTR-2115	

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts 01730

13. ABSTRACT

This report (1) identifies the features which distinguish SIMSCRIPT from general programming languages, permitting readers to judge for themselves the benefits of using SIMSCRIPT in their own applications; (2) outlines the language and implementation differences between the various versions of SIMSCRIPT; (3) specifies the resource requirements and relative advantages of implementing each version of SIMSCRIPT at MITRE/ESD; and (4) investigates the desirability of using SIMSCRIPT at ESD for analyzing problems related to computer performance.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
COMPUTERIZED SIMULATION						
COMPUTER PROGRAMMING						
COMPUTERS						
DIGITAL SIMULATION						
SIMULATION						
PROGRAMMING LANGUAGES						
PROGRAMMING MANUALS						